On the Storage and Retrieval of Continuous Media Data

Banu Özden Rajeev Rastogi Avi Silberschatz

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
{ozden, rastogi, avi}@research.att.com

Abstract

Continuous media applications, which require a guaranteed transfer rate of the data, are becoming an integral part of daily computational life. However, conventional file systems do not provide rate guarantees, and are therefore not suitable for the storage and retrieval of continuous media data (e.g., audio, video). To meet the demands of these new applications, *continuous media file systems*, which provide rate guarantees by managing critical storage resources such as memory and disks, must be designed.

In this paper, we highlight the issues in the storage and retrieval of continuous media data. We first present a simple scheme for concurrently retrieving multiple continuous media streams from disks. We then introduce a a clever allocation technique for storing continuous media data that eliminates disk latency and thus, drastically reduces RAM requirements. We present, for video data, schemes for implementing the operations fast-forward, rewind and pause. Finally, we conclude by outlining directions for future research in the storage and retrieval of continuous media data.

1 Introduction

The recent advances in compression techniques and broadband networking enable the use of continuous media applications such as multimedia electronic-mail, interactive TV, encyclopedia software, games, news, movies, on-demand tutorials, lectures, audio, video and hypermedia documents. These applications deliver to users continuous media data like video that is stored in digital form on secondary storage devices. Furthermore, continuous media-on-demand systems enable viewers to playback the media at any time and control the presentation by VCR like commands.

An important characteristic of continuous media that distinguishes it from non-continuous media (e.g., text) is that continuous media has certain timing characteristics associated with it. For example, video data is typically stored in units that are frames and must be delivered to viewers at a certain rate (which is typically 30 frames/sec). Another feature is that most continuous media types consume a large storage space and bandwidth. For example, a 100 minute movie compressed using the MPEG-I compression algorithm requires about 1.25 gigabyte (GB) of storage space. At a cost of 40 dollars per megabyte (MB), storing a movie in RAM would cost about 45,000 dollars. In comparison, the cost of storing data on disks is less than a dollar per megabyte and on tapes and CD-ROMs, it is of the order of a few cents per megabyte. Thus, it is more cost-effective to store video data on secondary storage devices like disks.

Given the limited amount of resources such as memory and disk bandwidth, it is a challenging problem to design a file system that can concurrently service a large number of both conventional and continuous media applications while providing low response time. Conventional file systems provide no rate guarantees for data retrieved and are thus unsuitable for the storage and retrieval of continuous media data. *Continuous media file systems*, on the other hand, guarantee that once a continuous media *stream* (that is, a request for the retrieval of a continuous media clip) is accepted, data for that stream is retrieved at the required rate.

The fact that the secondary storage devices have relatively high latencies and low transfer rates makes the problem more interesting. For example, besides the fact that disk bandwidths are relatively low, the disk latency imposes high buffering requirements in order to achieve a transfer rate close to the disk bandwidth. As a matter of fact, in order to support multiple streams, the closer the transfer rate gets to the disk bandwidth the higher the buffering requirements become. However, since the number of concurrent requests that can be serviced concurrently is dependent on both the buffering and bandwidth requirements, increasing transfer rate does not necessarily increase the number of requests that can be serviced concurrently.

inner track transfer rate	r_{disk}	68 Mb/sec
Settle time	t_{settle}	0.6 msec
Seek time (worst case)	t_{seek}	17 msec
Rotational latency (worst case)	t_{rot}	8.34 msec

Figure 1: The characteristics of Seagate Barracuda 2 disk.

In order to increase performance, schemes for reducing the impact of latency, as well as solutions for increasing bandwidth must be devised. Clever storage allocation schemes [2, 12, 3] as well as novel disk scheduling schemes [1, 11, 7, 13, 9] must be devised to reduce or totally eliminate latency so that buffering requirements can be reduced while bandwidth is utilized effectively. Storage techniques based on multiple disks such as replication and striping must be employed to increase the bandwidth.

In this paper, we first present a simple scheme for concurrently retrieving multiple continuous media streams from disks. We then show, how by employing novel striping techniques for storing continuous media data, we can completely eliminate disk latency and thus, drastically reduce RAM requirements. We present, for video data, schemes for implementing the basic VCR operations fast-forward, rewind, and pause. We conclude by outlining directions for future research in the storage and retrieval of continuous media data.

2 Retrieving Continuous Media Data

In this section, we briefly review characteristics of disks (additional details can be found in [14]), and present our architecture for retrieving continuous media streams from disks. We then outline a simple scheme for retrieving multiple concurrent continuous media streams from disks, and compute the buffer requirements of this scheme.

Data on disks is stored in a series of concentric circles, or *tracks*, and accessed using a disk head. A disk rotates on a central spindle and the speed of rotation denotes the transfer rate of the disk. Data on a particular track is accessed by positioning the head on (also referred to as *seeking* to) the track containing the data, and then waiting until the disk rotates enough so that the head is positioned directly above the data. Seeks typically consist of a coast during which the head moves at a constant speed and a settle, when the head position is adjusted to the desired track. Thus, the latency for accessing data on disk is the sum of seek and rotational latency. Another feature of disks is that tracks are longer at the outside than at the inside. A consequence of this is that outer tracks may have higher transfer rates than inner tracks. Figure 1 illustrates the notation we use for disk characteristics and the characteristics of the Seagate Barracuda 2 disk (we choose the disk transfer rate to be the transfer rate of the innermost track.)

In our architecture, we assume that continuous media clips are stored on disks and must be delivered at a rate r_{med}. The continuous media system is responsible for retrieving data for continuous media streams from disk into RAM at rate r_{med}. The data is then transmitted over a network to clients where they are delivered at the required rate. In this paper, we restrict ourselves to the problem at the server end – that is, the task of retrieving multiple continuous media streams from disk to RAM concurrently.

The maximum number of concurrent streams, denoted by p, that can be retrieved from disk is given by

$$p = \lfloor \frac{r_{disk}}{r_{med}} \rfloor. \qquad (1)$$

A simple scheme for retrieving data for m continuous media streams concurrently is as follows. Continuous media clips are stored contiguously on disk and a buffer of size d is maintained in RAM for each of the m streams. Continuous media data is retrieved into each of the buffers at a rate r_{med} in a round robin fashion, the number of bits retrieved into a buffer during each round being d. In order to ensure that data for the m streams can be continually retrieved from disk at a rate r_{med}, in the time that the d bits from m buffers are consumed at a rate r_{med}, the d bits following the d bits consumed must be retrieved into the buffers for every one of the m streams. Since each retrieval involves positioning the disk head at the desired location and then transferring the d bits from the disk to the buffer, we have the following equation.

$$\frac{d}{r_{med}} \geq m \cdot (\frac{d}{r_{disk}} + t_{seek} + t_{rot})$$

In the above equation, $\frac{d}{r_{disk}}$ is the time it takes to transfer d bits from disk, and $t_{seek} + t_{rot}$ is the worst case disk latency. Hence, the size d of the buffer per stream can be calculated as

$$d \geq \frac{(t_{seek} + t_{rot}) \cdot r_{med} \cdot r_{disk}}{(\frac{r_{disk}}{m} - r_{med})}. \qquad (2)$$

Thus, the buffer size per stream increases both with latency of the disk and the number of concurrent streams. In the following example, we compute for a commercially available disk, the buffer requirements in order to support the maximum number of concurrent streams.

Example 1: Consider MPEG-I compressed video data stored on a Seagate Baracuda 2 disk. The video data needs to be retrieved at a rate of r_{med} =1.5 Mb/s. Thus, the maximum number of streams that can be retrieved from the disk is 45. Since the worst-case rotational latency of 8.34 msec and worst case seek latency of 17 msec, the worst-case latency for the disk is 25.34 msec. From Equation 2, it follows that the minimum buffer size required in order to support 45 streams is 233 Mb. Since there is a buffer of size d for every stream, the total buffer requirements are 10 Gb. □

In the case of video streams, the VCR operations pause, fast-forward, and rewind can be implemented as follows. Pause is implemented by simply halting the consumption of bits from the buffer for the stream. Furthermore, the number of bits, d_1, read into the buffer during a round satisfies the following equality.

$$d_1 + d_2 = d$$

where d_2 is the number of unconsumed bits already contained in the buffer before data is read into it. Thus, it is possible that when a stream is paused, no data is read into the buffer for the stream until it is resumed again. Fast-forward is implemented by simply skipping a certain number of bits in the continuous media clip between the d bits retrieved during each successive round into the buffer for the stream. Similarly, rewind is implemented by retrieving preceding bits during each successive round, and skipping a certain number of bits between the d bits retrieved during successive rounds.

3 Matrix-Based Allocation

The scheme we proposed in Section 2 for retrieving data for multiple continuous media streams had high buffer requirements due to high disk latencies. In this section, we present a clever storage allocation scheme for video clips that completely eliminates disk latency and thus, keeps buffer requirements low. However, the scheme results in an increase in the worst-case response time between the time a request for a continuous media clip is made and the time the data for the stream can actually be consumed.

3.1 Storage Allocation

In order to keep the amount of buffer required low, we propose a new storage allocation scheme for continuous media clips on disk, which we call the *matrix-based* allocation scheme. This scheme is referred to as phase-constrained allocation in [2] when it is used to store a single clip. The matrix-based allocation scheme eliminates seeks to random locations, and thereby enables the concurrent retrieval of maximum number of streams p, while maintaining the buffer requirements as a constant independent of the number of streams and disk latencies. Since continuous media data is retrieved sequentially from disk, the response time for the initiation of a continuous media stream is high.

Consider a *super-clip* in which the various continuous media clips are arranged linearly one after another. Let l denote the length of the super-clip in seconds. Thus, the storage required for the super-clip is $l \cdot r_{med}$ bits. Suppose that continuous media data is read from disks in portions of size d. We shall assume that $l \cdot r_{med}$ is a multiple of $p \cdot d$.[1] Our goal is to be able to support p concurrent continuous media streams. In order to accomplish this, we divide the super-clip into

[1]The length of the super-clip can be modified by appending advertisements, etc. to the end of the super-clip.

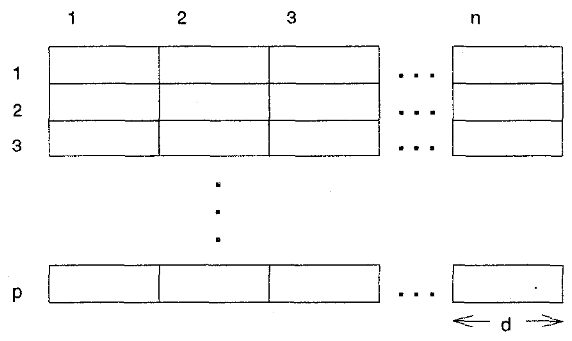

Figure 2: The super-clip viewed as a matrix.

p contiguous partitions. Thus, the super-clip can be visualized as a $(p \times 1)$ vector, the concatenation of whose rows is the super-clip itself and each row contains $t_c \cdot r_{med}$ bits of continuous media data, where

$$t_c = \frac{l}{p}.$$

Note that the first bit in any two adjacent rows are t_c seconds apart in the super-clip. Also, a continuous media clip in the super-clip may span multiple rows. Since super-clip data in each row is retrieved in portions of size d, a row can be further viewed as consisting of n portions of size d, where

$$n = \frac{t_c \cdot r_{med}}{d}$$

Thus, the super-clip can be represented as a $(p \times n)$ matrix of portions as shown in Figure 2. Each portion in the matrix can be uniquely identified by the row and column to which it belongs. Suppose we now store the super-clip matrix on disk sequentially in column-major form. Thus, as shown in Figure 3, Column 1 is stored first, followed by Column 2, and finally Column n.

We now show that by sequentially reading from disk, the super-clip data in each row can be retrieved concurrently at a rate r_{med}. From Equation 1, it follows that:

$$\frac{p \cdot d}{r_{disk}} \leq \frac{d}{r_{med}}, \tag{3}$$

Therefore, in the time required to consume d bits of continuous media data at a rate r_{med}, an entire column can be retrieved from disk. As a result, while a portion is being consumed at a rate r_{med}, the next portion can be retrieved.

Suppose that once the n^{th} column has been retrieved, the disk head can be repositioned to the start of the device almost instantaneously. In this case, we can show that p concurrent streams can be supported while the worst case response time for the initiation of a stream will be t_c. The reason for this is that every t_c seconds the disk head can be repositioned to the start. Thus, the same portion of a continuous media clip is retrieved every t_c seconds. Furthermore, for every other concurrent stream, the last portion retrieved just before the disk head is repositioned, belongs to Column n. Since we assume that repositioning time is negligible, Column 1 can be retrieved immediately after Column n. Thus, since the portion following portion (i,n) in Column n, is portion $(i+1,1)$ in Column 1, data for concurrent streams can be retrieved from disk at a rate r_{med}. In Section 3.3, we present schemes that take into account repositioning time when retrieving data for p concurrent streams.

3.2 Buffering

We now compute the buffering requirements for our storage scheme. Unlike the scheme presented in Section 2 in which we associated a buffer with every stream, in the matrix-based scheme, with every row of the super-clip matrix, we associate a *row buffer*, into which consecutive portions in the row are retrieved. Each of the row buffers is implemented as a *circular* buffer; that is, while writing into the buffer, if the end is reached, then further bits are written at the beginning of the row buffer (similarly, while reading, if the end is reached, then subsequent bits are read from the beginning of the buffer).

With the above circular storage scheme, every $\frac{t_c}{n}$ seconds, consecutive columns of the super-clip data are retrieved from disk into row buffers. The size of each buffer is $2 \cdot d$, one half of which is used to read in a portion of the super-clip from disk, while d bits of the super-clip are consumed from the other half. Also, the number of row buffers is p. The row buffers store the p different portions of the super-clip contained in a single column – the first portion in a column is read into the first row buffer, the second portion into the second row buffer and so on. Thus, in the scheme, initially, the p portions of the super-clip in the first column are read into the first d bits of each of the corresponding row buffers. Following this, the next p portions in the second column are read into the latter d bits of each of the corresponding row buffers. Concurrently, the first d bits from each of the row buffers can be consumed for the p concurrent streams. Once the portions from the second column have been retrieved, the portions from the third column are retrieved into the first d bits of the row buffers and so on. Since consecutive portions of a super-clip are retrieved every $\frac{t_c}{n}$ seconds, consecutive portions of continuous media clips in the super-clip are retrieved into the buffer at a rate of r_{med}. Thus, in the first row buffer, the first n portions of the super-clip (from the first row) are output at a rate of r_{med}, while in the second, the next n portions (from the second row) are output and so on. As a result, a request for a continuous media stream can be initiated once the first portion of the continuous media clip is read into a row buffer. Furthermore, in the case that a continuous media clip spans multiple rows, data for the stream can be retrieved by sequentially accessing the contents of consecutive row buffers.

3.3 Repositioning

The storage technique we have presented thus far enables data to be retrieved continuously at a rate of r_{med} under the assumption that once the n^{th} column of the super-clip is retrieved from disk, the disk head can be repositioned at the start almost instantaneously. However, in practice, this assumption does not hold. Below, we present techniques for retrieving data for p concurrent streams of the super-clip if we were to relax this assumption. The basic problem is to retrieve data from the device at a rate of r_{med} in light of the fact that no data can be transferred while the head is being repositioned at the start. A simple solution to this problem is to maintain another disk which stores the super-clip exactly as stored by the first disk and which takes over the function of the disk while its head is being repositioned.

An alternate scheme, which does not require the entire super-clip to be duplicated on both disks, can be employed if t_c is at least twice the repositioning time. The super-clip data matrix is divided into two submatrices so that one submatrix contains the first $\lceil \frac{n}{2} \rceil$ columns and the other submatrix, the remaining $\lfloor \frac{n}{2} \rfloor$ columns of the original matrix, and each submatrix is stored in column-major form on two disks with bandwidth r_{disk}. The first submatrix is retrieved from the first disk, and then the second submatrix is read from the other disk while the first disk is repositioned. When the end of the data on the second disk is reached, the data is read from the first disk and the second disk is repositioned.

If the time it takes to reposition the disk to the start is low, in comparison to the time it takes to read the entire super-clip, as is the case for disks, then almost at any given instant one of the disks would be idle. To remedy this deficiency, in the following, we present a scheme that is more suitable for disks. In the scheme, we eliminate the additional disk by storing, for some m, the last m portions of the column-major form representation of the super-clip in RAM so that after the first $lr_{med} - md$ portions have been retrieved from the disk into the row buffers, repositioning of the head to the start is initiated. Furthermore, while the device is being repositioned, the last m portions of the super-clip are retrieved into the row buffers from RAM instead of the device. Once the head is repositioned and the last m portions have been retrieved into the row buffers, the columns are once again loaded into the row buffers from disk beginning with the first column as described earlier in the section. For the above scheme to retrieve data for streams at a rate of r_{med}, the time to reposition the head must be less than or equal to the time to consume m portions of continuous media data at a rate of r_{med}, that is,

$$\frac{m \cdot d}{r_{med}} \geq t_{seek} + t_{rot}$$

Thus, the total RAM required is $md + 2dp$.

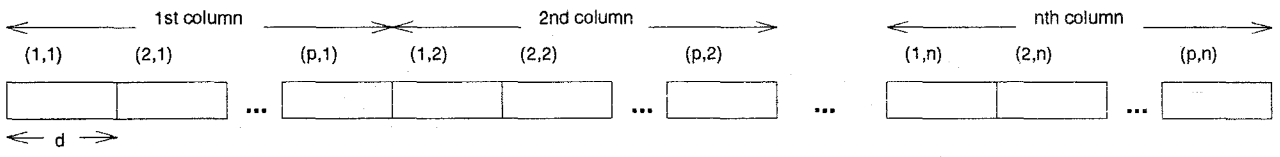

Figure 3: Placement of n columns of the super-clip matrix.

3.4 Implementation of VCR Operations

We now describe how the VCR operations begin, pause, fast-forward, rewind and resume can be implemented with the matrix-based storage architecture. As we described earlier, contiguous portions of the super-clip are retrieved into p row buffers at a rate r_{med}. The first n portions are retrieved into the first row buffer, the next n into the second row buffer, and so on.

- **begin:** The consumption of bits for a continuous media stream is initiated once a row buffer contains the first portion of the continuous media clip. Portions of size d are consumed at a rate r_{med} from the row buffer (wrapping around if necessary). After the $i \cdot n^{th}$ portion of the super-clip is consumed by a stream, consumption of data by the stream is resumed from the $i+1^{th}$ row buffer. We refer to the row buffer that outputs the continuous media data currently being consumed by a stream as the *current* row buffer. Since in the worst case, $n \cdot d$ bits may need to be transmitted before a row buffer contains the first portion of the requested continuous media clip, the delay involved in the initiation of a stream when a begin command is issued, in the worst case, is t_c.
- **pause:** Consumption of continuous media data by the stream from the current row buffer is stopped (note however, that data is still retrieved into the row buffer as before).
- **fast-forward:** A certain number of bits are consumed from each successive row buffer following the current row buffer. Thus, during fast-forward, the number of bits skipped between consecutive bits consumed is approximately $n \cdot d$ (note that this scheme is inapplicable if successive row buffers do not contain data belonging to the same continuous media clip).
- **rewind:** This operation is implemented in a similar fashion to the fast-forward operation except that instead of jumping ahead to the following row buffer, jumps during consumption are made to the preceding row buffer. Thus, a certain number of bits are consumed from each previous row buffer preceding the current row buffer.
- **resume:** In case the previously issued command was either fast-forward or rewind, bits are continued to be consumed normally from the current row buffer. If, however, the previous command was pause, then once the current row buffer contains the bit following the last bit consumed, normal consumption of data from the row buffer is resumed beginning with the bit. Thus, in the worst case, similar to the case of the begin operation, a delay of t_c seconds may result before consumption of data for a stream can be resumed after a pause operation.

For the disk in Example 1, t_c for a 100 minute super-clip is approximately 133 seconds. Thus, the worst case delay is 133 seconds when beginning or resuming a continuous media stream. Furthermore, the number of frames skipped when fast-forwarding and rewinding is 3990 (133 seconds of video at 30 frames/s). By reducing t_c, we could reduce the worst-case response time when initiating a stream.

We now show how multiple disks can be employed to reduce t_c. Returning to Example 1, suppose that instead of using a single disk, we were to use an array of 5 disks. In this case, the bandwidth of the disk array increases from 68 Mb/s to 340 Mb/s. The number of streams, p, increases from 45 to 226, and, therefore, t_c reduces from 133 seconds to approximately 26 seconds. In this system, the worst case delay is 26 seconds and the number of frames skipped is 780 (26 seconds of video at 30 frames/sec).

4 Related Work

A number of storage schemes for continuous retrieval of video and audio data have been proposed in the literature [5, 11, 12, 7, 3, 8, 13, 9]. Among these, [5, 11, 7, 13, 9] address the problem of satisfying multiple concurrent requests for the retrieval of multimedia objects residing on a disk. These schemes are similar in spirit to the simple scheme that we presented in Section 2. In each of the schemes, concurrent requests are serviced in rounds retrieving successive portions of multimedia objects and performing multiple seeks in each round. *Admission control* tests based on computed buffer requirements for multiple requests are employed in order to determine the feasibility of additional requests with available resources. The schemes presented in [5, 11, 7] do not attempt to reduce disk latency. In [13], the authors show that the CSCAN policy for disk scheduling is superior for retrieving continuous media data in comparison to a policy in which requests with the earliest deadlines are serviced first (EDF) [10]. In [9], the authors propose a greedy disk scheduling algorithm in order to reduce both seek time and rotational latency.

In [3], in order to reduce buffer requirements, an audio record is stored on optical disk as a sequence of data blocks separated by gaps. Furthermore, in order to save disk space, the authors derive conditions for merging different audio records. In [12], similar to [3], the authors define an interleaved storage organization for multimedia data that permits the merging of time-dependent multimedia objects for efficient disk space utilization. However, they adopt a weaker condition for merging different media strands, a consequence of which is an increase in the read-ahead and buffering requirements.

In [8], the authors use parallelism in order to support the display of high resolution of video data that have high bandwidth requirements. In order to make up for the low I/O bandwidths of current disk technology, a multimedia object is declustered across several disk drives, and the aggregate bandwidth of multiple disks is utilized.

5 Research Issues

In this section, we discuss some of the research issues in the area of storage and retrieval of continuous media data that remain to be addressed.

5.1 Load Balancing and Fault Tolerance Issues

So far, we assumed that continuous media clips are stored on a single disk. However, in general, continuous media servers may have multiple disks on which continuous media clips may need to be stored. One approach to the problem is to simply partition the set of continuous media clips among the various disks and then use the schemes that we described earlier in order to store the clips on each of the disks. One problem with the approach is that if requests for continuous media clips are not distributed uniformly across the disks, then certain disks may end up idling, while others may have too much load and so some requests may not be accepted. For example, if clip C_1 is stored on disk D_1 and clip C_2 is stored on disk D_2, then if there are more requests for C_1 and fewer for C_2, then the bandwidth of disk D_2 would not be fully utilized. A solution to this problem is *striping*. By storing the first half of C_1 and C_2 on D_1 and the second half of the clips on D_2, we can ensure that the workload is evenly distributed between D_1 and D_2.

Striping continuous media clips across disks involves a number of research issues. One is the granularity of striping for the various clips. The other is that striping complicates the implementation of VCR operations. For example, consider a scenario in which every stream is paused just before data for the stream is to be retrieved from a "certain" disk D_1. If all the streams were to be resumed simultaneously, then the resumption of the last stream for which data is retrieved from D_1 may be delayed by an unacceptable amount of time. Replicating the continuous media clips across multiple disks could help in balancing the load on disks as well as reducing response times in case disks get overloaded.

Replication of the clips across disks is also useful to achieve fault-tolerance in case disks fail. One option is to use disk mirroring to recover from disk failures; another would be to use parity disks [6]. The potential problem with both of these approaches is that they are wasteful in both storage space as well as bandwidth. We need alternative schemes that effectively utilize disk bandwidth, and at the same time ensure that data for a stream can continue to be retrieved at the required rate in case of a disk failure. Finally, an interesting research issue is to vary the size of buffers allocated for streams dynamically, based on the number of streams being concurrently retrieved from disk at any point in time.

5.2 Storage Issues

Neither storing clips contiguously, nor the matrix-based storage scheme for continuous media clips is suitable in case there are frequent additions, deletions and modifications. The reason for this is that both schemes are very rigid and could lead to fragmentation. As a result, we need to consider storage schemes that decompose the storage space on disks into pages and then map various continuous media clips to a sequence of non-contiguous pages. Even though the above scheme would reduce fragmentation, since pages containing a clip may be distributed randomly across the disk, disk latency would increase resulting in increased buffer requirements. An important research issue is to determine the ideal page size for clips that would keep both space utilization high as well as disk latency low.

Another important issue to consider is the storage of continuous media clips on tertiary storage (e.g., tapes, CD-ROMs). Since continuous media data tends to be voluminous, it may be necessary (in order to reduce costs) to store it on CD-ROMs and tapes, which are much cheaper than disks. Techniques for retrieving continuous media data from tertiary storage is an interesting and challenging problem. For example, tapes have high seek times and so we may wish to use disks to cache initial portions of clips in order to keep response times low.

5.3 Data Retrieval Issues

One of the schemes we presented in the previous sections yields low response times but has significantly large buffer requirements. The other scheme, on the other hand, has much higher response times but it eliminates disk latency completely and has low buffer requirements. Further research along the lines of [1, 13, 9] must be carried out in order to reduce disk seek and rotational latency when servicing stream requests while keeping response times low.

We have also assumed so far that media streams have a single transfer rate r_{med}. This assumption may not be true. For example MPEG-I compressed video requires a transfer rate of 1.5 Mb/s, while JPEG compressed video requires a transfer rate of about 7 Mb/s [4]. We need to develop schemes to retrieve data for continuous media streams with different transfer rates.

In the schemes we developed, we made the pessimistic assumption that the disk transfer rate r_{disk} is equal to the transfer rate of the innermost track. By taking into account the disk transfer rate of the tracks where continuous media clips are stored, we could substantially reduce buffer requirements. Also, in our work, we have not taken into account the fact that disks are not perfect. For example, disks have bad sectors that are remapped to alternate locations. Furthermore, due to thermal expansion, tables storing information on how long and how much power to apply on a particular seek, need to be recalibrated. Typically, this takes 500-800 milliseconds and occur once every 15-30 minutes. Finally, in this paper, we have only considered continuous media requests. We need a general-purpose system that would have the ability to service both continuous (e.g., video, audio) as well as non-continuous (e.g., text) media requests. Such a system would have to give a high priority to retrieving continuous media data, and use *slack* time in order to service non-continuous media requests.

6 Concluding Remarks

In this paper, we considered two approaches to retrieving continuous media data from disks. In the first approach, response times for servicing requests are low, but a high latency is incurred, resulting in significantly large buffer requirements. The second approach eliminates random disk head seeks and thus reduces buffer requirements but may result in increased response times. We presented simple schemes for implementing pause, fast-forward and rewind operations on continuous media streams (in case the data is video data). Finally, we outlined future research issues in the storage and retrieval of continuous media data.

References

[1] B. Özden, R. Rastogi and A. Silberschatz. $\mathcal{F}$ellini —a file system for continuous media. Technical report, AT&T Bell Laboratories, June 1994.

[2] B. Özden, R. Rastogi, A. Biliris and A. Silberschatz. A low-cost storage server for movie on demand databases. In *Proceedings of the Twentieth International Conference on Very Large Databases, Santiago*, September 1994.

[3] D. Bitton, Q. Yang, R. Bruno, C. Yu, W. Sun and J. Tullis. Efficient placement of audio data on optical disks for real-time applications. *Communications of the ACM*, 32(7):862–871, July 1989.

[4] H. J. Chen and T. D. C. Little. Physical storage organizations for time-dependent data. In *Foundations of Data Organization and Algorithms*, pages 19–34. Springer-Verlag, October 1993.

[5] Y. Osawa, D. P. Anderson and R. Govindan. A file system for continius media. *ACM Transactions on Computer Systems*, 10(4):311–337, November 1992.

[6] R. Y. Hou, G. R. Ganger, B. L. Worthington and Y. N. Patt. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):564–573, August 1993.

[7] J. Gemmell and S. Christodoulakis. Principles of delay-sensitive multimedia data storage and retreival. *ACM Transactions on Information Systems*, 10(1):51–90, January 1992.

[8] S. Ghandeharizadeh and L. Ramos. Continuous retrieval of multimedia data using parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):658–669, August 1993.

[9] A. Goyal, H. M. Vin and P. Goyal. Algorithms for designing large-scale multimedia servers. *Computer Communication*, 1994.

[10] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in a har real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[11] H. M. Vin, P. V. Rangan and S. Ramanathan. Designing an on-demand multimedia service. *IEEE Communications Magazine*, 1(1):56–64, July 1992.

[12] P. V. Rangan and H. M. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):564–573, August 1993.

[13] A. L. N. Reddy and J. C. Wyllie. I/O issues in a multimedia system. *Computer*, 27(3):69–74, March 1994.

[14] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *Computer*, 27(3):17–27, March 1994.